Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 85 (2016) 598 – 605

Procedia Computer Science

International Conference on Computational Modeling and Security (CMS 2016)

Data-local Reduce Task Scheduling

Geetha J[a], N UdayBhaskar [a,]*, P ChennaReddy[a,]*

[a]*Department of Computer Science and Engineering, M S Ramaiah Institute of Technology, Bangalore, 560054, India*

Abstract

Inspired by the victory of Apache's Hadoop this paper suggests a new reduce task scheduler. Hadoop is an open source implementation of Google's MapReduce framework. Programs which are written in this functional style are automatically executed and parallelized on a large cluster of commodity machines. The details how to partition the input data, setting up the program's for execution across a set of machines, handling failures of machine and managing the necessary inter-device communication is taken care by runtime system. In the current versions of Hadoop, the map tasks are scheduled with respect to the locality of their inputs in order to shrink network traffic and improve performance. On the other hand, the reduce tasks are scheduled without taking into consideration data locality leading to ruin the performance at requesting nodes. In this paper, we use data locality that is natural with reduce tasks. To accomplish the same, we schedule them on nodes that will result in least amount data- local traffic. Experimental results signify an 11-80 percent decrease in the number of bytes shuffled in a Hadoop cluster**.**

Peer-review under responsibility of the Organizing Committee of CMS 2016

Keywords: MapReduce, Hadoop , Reduce Task Scheduling , Data-Locality , Rack-Locality.

1. Introduction

Every day, the Stock Exchange generates every day about one terabyte of new trade data [3]. Facebook generates 5 Billion terabytes of data every day. Any such data that cannot be placed into a database falls under the category of unstructured data. As an entity's data footprint grows, the amount of unstructured data to be handled becomes enormous. This is a major cause for several problems .First, where will all this Big Data, as it is being calledtoday, be stored. Second, how do we access all this data, process and analyze the contents. Third, how do we ensure that this data is safe from disk failures, computer failures and so on?

These problems are well-dealt by Hadoop; a reliable, scalable, distributed computing platform developed by

*Corresponding author. Tel.:+919885238374
E-mail address: udaynagella@gmail.com, pcreddy1@rediffmail.com

Peer-review under responsibility of the Organizing Committee of CMS 2016
doi:10.1016/j.procs.2016.05.226

Apache [5].It is an open source implementation of Google's MapReduce framework that allows for the distributed Processing of huge data sets transversely clusters of computers using simple programming models.It is designed to level up from single datacenter to thousands of machines, each offering local computation and storage. At the application layer the library is designed to detect and handle failures, Instead of relying on hardware to deliver high-availability, so a highly-available service on top of a cluster of computers can be delivered each of which may be prone to failures Page Layout.

Hadoop has an underlying storage system called HDFS-Hadoop Distributed file system. To process the data in HDFS, Ha doop provides a MapReduce engine that runs on top of HDFS. This engine has master-slave architecture. The master node is called the JobTracker and the slaves are called TaskTrackers. MapReduce jobs are automatically parallelized across a large set of TaskTrackers. The JobTracker splits the job into several maps and reduce tasks. Hadoop divides the input to the job into fixed size pieces called input splits. The outputs of the map tasks are stored in the local disk. This intermediate output serves as input to the reduce tasks. The consolidated output of the reduce task is the output of the MapReduce job and is stored in HDFS.

Recently, the non-profit organization Spamhaus, which publishes spam blacklists, faced one of the most powerful cyber-attacks seen. This led to cyberspace congestion which affected the Internet overall. The providers of Spamhaus as well as certain Tier 1 Internet Exchanges were the victims of the attack. Cloudflare dubbed it as the attack "that almost broke the Internet." These incidents do highlight the need to develop more efficient security measure to combat such attacks but it also indicates how uncontrolled network congestion can handicap the Internet as a whole. Measures must be taken to tackle any source of congestion within a network. With companies like Facebook, Amazon, AOL and The NY Times using the Hadoop framework which in turn is a cluster of computers connected via a LAN or MAN connection, there is a need to handle any areas that could result in network traffic and hence result in significant delay in providing services to the end user.

A typical Hadoop cluster is a set of computers connected via LAN connection. In case of companies like Facebook, Yahoo, Amazon these clusters consist of thousands of nodes. These Hadoop workers are located in different data centres that are present in geographically dispersed locations thereby avoiding the domino effect that is prevalent when all nodes are connected by a single LAN. A typical MapReduce job may use nodes across data centres .Each node has a local disk, it is efficient to move data processing operations to nodes where application data are located. If data is not available locally in a processing node, data has to be migrate using network interconnects to the node that performs the operation of data processing s. Migrating huge amount of data across data centers and in some cases within data centers may lead to excessive network congestion especially when petabytes of data have to be processed.

In Hadoop, scheduling reduce tasks results in severe congestion problems. The reason being Hadoop task scheduling is based on a 'pull strategy'. The JobTracker does not push map and reduce tasks to TaskTrackers but rather TaskTrackers pull them by making requests. Every TaskTrackers sends a periodic heart beat message requesting a map or reduce tasks. When the JobTracker schedules map tasks, it takes care to ensure that the task runs on a TaskTracker which contains the needed input split. As a result Hadoop MapReduce is said to be data local when scheduling map tasks. On the other hand Hadoop simply schedules any yet-to-run reduce task on any requesting TaskTracker regardless of the TaskTracker's network location [3].This has an adverse effect on network traffic .

This paper proposes scheduling data local reduce tasks. The method ensures that reduce tasks run on a suitable TaskTracker such that there is minimum bytes travelling across the network. Thus, there is a controlled use of the clusters network bandwidth. We have implemented our method on Hadoop 1.0.3.

The rest of this paper is organized as follows. A background on Hadoop MapReduce is given in Section 2. Our proposal and implementation is presented in Section 3, and a performance evaluation in Section 4.Lastly we indicate future works and conclude the paper in Section 5.

.

2. Background : Reduce Task Scheduler

Hadoop assumes a tree-style network topology similar to the one shown in the figure. Nodes are distributed over different racks contained in one or many data centers. A salient point is that the bandwidth between two nodes is dependent on their relative locations in the network topology.

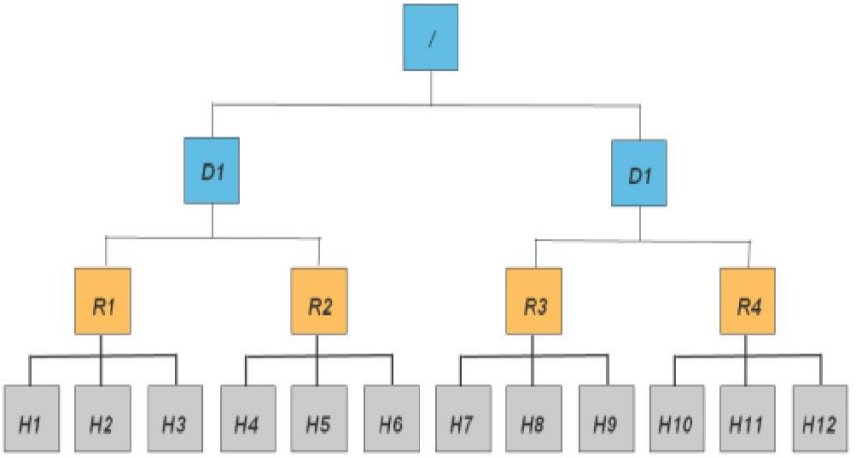

Fig1: *Network Topology in Hadoop (D= data center, R = rack, and H = node)*

For example, nodes that are on the same rack will have high bandwidth between them as contrasting to nodes that are off-rack [17]. Previous attempts to make reduce tasks locality aware focused on rack-locality not data-locality. The tasks were scheduled in racks that held the largest intermediate output. However, this does not address the problem when using zero or one rack or the network traffic that is prevalent when scheduling reduce tasks within a single data centre. There is need to penetrate deeper into the network hierarchy. In other words, optimization so far to make reduce tasks locality aware, addresses scenarios with many nodes connected via racks. The approach proposed in this paper further optimizes scheduling reduce tasks in small, medium and large clusters. Further, it works in the presence or absence of racks. It addresses the problem when there are two nodes and also when there are innumerable nodes.

3. The Data Local Reduce Task Scheduler

3.1. Motivation

Consider the scenario as shown in Fig.1. Node 1 holds 5 MB IO: R1 (intermediate output for reducer 1) and 10 MB IO: R2 (intermediate output for reducer 2). While Node 2 holds 2 MB IO: R1 and 26 MB IO: R2. Node 3 holds 15 MB IO: R1 and 1MB IO: R2 .Once the map tasks are scheduled, Node 2 requests the JobTracker for permission to run reducer 1. As a result 20 MB of intermediate output must be transferred over the network. On the other hand node 1 requests to run reducer 2.As a result 27 MB must be moved to node 1. Hence, a total of 47 MB utilizes the cluster bandwidth. If node 1 or node 2 is situated in different racks compared to other nodes more congestion in the network will be prevalent. If this scenario were modified, such that reducer 1 ran on node 3 and

reducer 2 ran on node 2, then 7MB and 11MB of data would be shuffled. Clearly, this result in an improvement of 50 % reduction in the number of bytes shuffled. Hadoop's present reduce task scheduler is incapable of making such decisions.

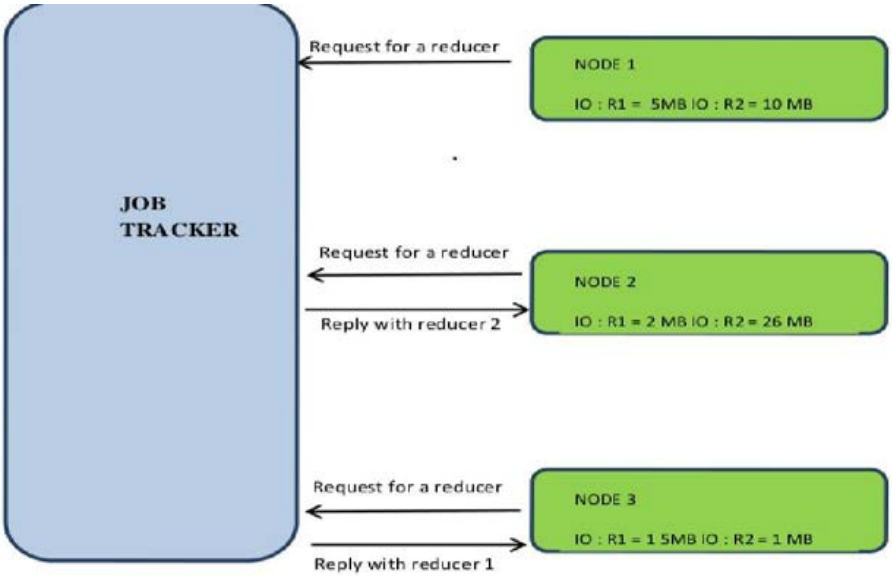

Fig 2: Scheduling Reduce Tasks in native Hadoop

3.2 Implementation

We have implemented our method on Hadoop 1.0.3 as follows:

- Formulation of a data structure that keeps track of the size of intermediate output generated by a mapper for every reducer - There are two in-built features of Hadoop that are exploited to gather the input required by the reduce task scheduler: *Index File and Heartbeat protocol.* The size of the intermediate output generated by a map task is available in the current versions of Hadoop. However, the manner in which the intermediate output is intended to be divided among the reducers is necessary. This information is available in the index file and is stored in the local file system of every TaskTracker. Heartbeat is a mechanism for a TaskTracker to announce periodic availability to the JobTracker. Besides the heartbeat, the TaskTrackers send information regarding its states to the JobTracker. By making modifications in appropriate classes, the JobTracker collects the index file information from the local file system of the TaskTracker via the information sent along with the heartbeat message.

- Controlling the scheduling of reduce tasks - Using the generated data structure, the TaskTracker that holds the largest intermediate output for a particular reduce task is determined. Upon which the reduce task scheduler takes the requesting TaskTracker as an input along with a set of unscheduled reduce tasks. For each reduce task that must be scheduled, the scheduler checks if the requesting TaskTracker is the one that contains the largest intermediate output. If so, the reduce task is scheduled on the requesting TaskTracker. Otherwise, another requesting TaskTracker is considered.

Algorithm 1 Reduce Task Scheduler

Input: RT: set of unscheduled reduce tasks
TT: the task tracker requesting a reduce task
STT: set of task trackers
Output: A reduce task R ∈ RT that can be scheduled at TT or -1

```
1.  for every reducer R ∈ RT do
2.   large = 0
3.   for every T ∈ STT do
4.      size = the size of the intermediate output on T
5.       if size > large then
6.       large = size
7.       LTT = T
8.       end if
9.   end for
10. end for
11. for the first reducer R ∈ RT do
12.  LTT =  the task tracker that holds the largest intermediate output
13.   if TT = LTT then
14.   return R
15.    else
16.   return -1(Request with another task tracker)
17. end for
```

3.3 Working Example

The JobTracker maintains a record that ensures that a reducer runs only on TaskTrackers that hold the largest intermediate output. The scenario mentioned previously is modified as shown in fig.3. In this case, to minimize data local traffic by a significant amount, the JobTracker analyzes and concludes that reducer 1 and reducer 2 must run on node 3 and node 2 respectively. The JobTracker initially is required to run reducer 1. The request from TaskTracker node 1 is rejected. Similarly, the request from TaskTracker node 2 is also overruled. Upon receiving a request from node 3, the JobTracker schedules the execution of reducer 1. Likewise, in the case of reducer 2, the requests of TaskTracker node 1 and node 3 are vetoed. The JobTracker schedules the execution of reducer 2 on node 2 upon receiving a request from the latter.

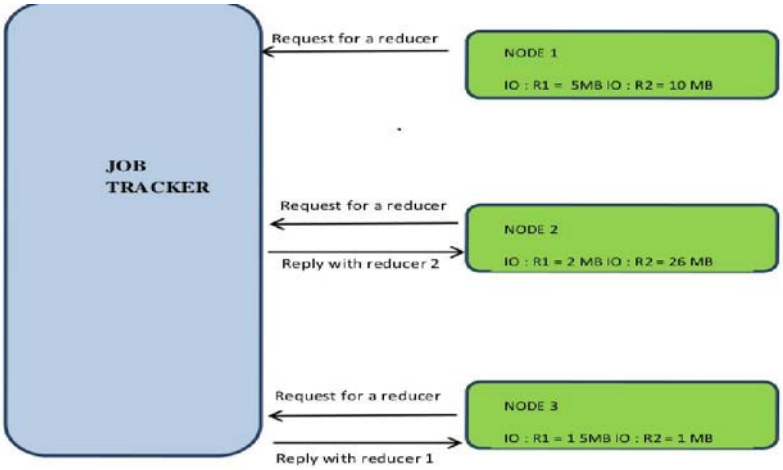

Fig 3: Achieving Data-Local Reduce Tasks

4. Performance Evaluation

All evaluations are performed on a four node cluster. Each node is a Intel® Core™2 Duo E8500 processor-based desktop. To run Hadoop, Ubuntu 12.10 and Sun/Oracle JDK 1.6, Update 20 has been installed on all computers.These nodes are connected by a 100Mbps Ethernet through a single switch. Three benchmarks are used for comparison with Native Hadoop .They are wordcount, pi estimator and muti-file wordcount. Besides, [10] reported that wordcount is one of the major benchmarks utilized for evaluating Hadoop at Yahoo. The wordcount is a CPU intensive job .It generates adequate amount of intermediate output .Multi-file count is another variation of word count that increases the workload and processes several files. Pi Estimator is suitable for testing the Hadoop scheduler when it deals with 1 KB data or lesser.It is a scenario with one reducer. This benchmark indicates that the modified Hadoop scheduler is capable of handling traffic prevalent within a single data center or a small cluster.

The native Hadoop scheduler is capable of making any reduce task scheduling decisions. As a result, large amount of network bandwidth and MapReduce execution time were lost trying to transfer huge amounts of data within and across networks. The difference in the number of bytes shuffled varies with the size of the intermediate output generated, the number of mappers and the number of reducers. Each benchmark is run three times as shown in Fig.4.

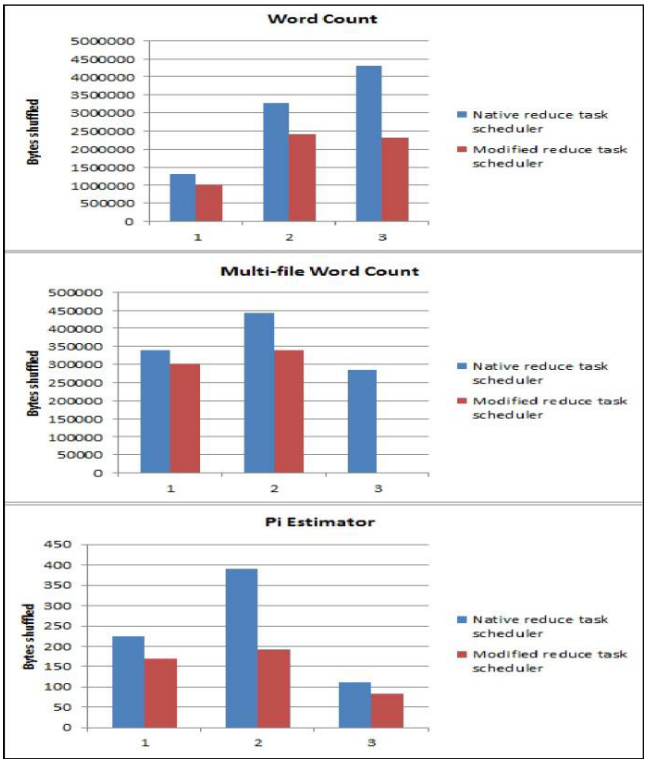

Fig 4: Variations in the number of bytes shuffled for three different benchmarks

In each case, the number of mappers and reducers are changed. The evaluation is performed on input size varying between120 bytes to 500 MB. The evaluation is performed when the number of mappers varies between 1 and 6. The number of reducers varies between 1 and 8. Further, as shown in Fig.5, the reduction in bytes shuffled grows with an increase in intermediate output and as a result with increase in the number of bytes input to the MapReduce program .The fluctuation is due to varying number of mappers and reducers. The modified Hadoop scheduler minimizes data-local traffic by 11-80 %.

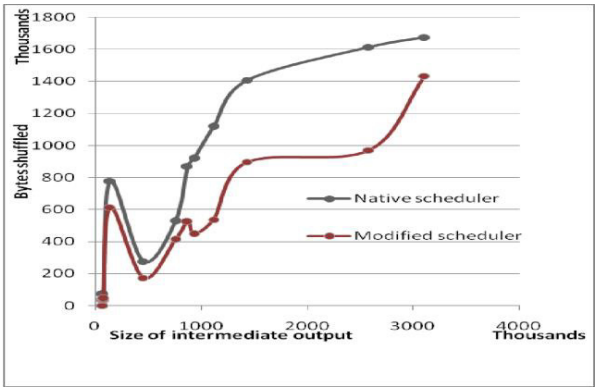

Fig 5: Variations in the number of bytes shuffled for three different benchmarks

5. Conclusion and Future Works

Through this paper, we have proposed a reduce task scheduler that can be incorporated in future versions of Hadoop. Unlike previous attempts, this reduce task scheduler is not limited to achieving rack locality alone. Further, it ensures data local execution of reduce tasks in scenarios with less than four nodes besides cases with nodes geographically distributed across data centres. Although, in circumstances with few bytes of input data the reduce task scheduler may appear to increase the execution time of a MapReduce job. However, this trend does not continue on increasing the number of bytes Hadoop operates on. Test results indicate a reduction of 11-80% in data local traffic.

The most important business benefits of including the proposed reduce task scheduler in Hadoop is lesser failure rates. This results in faster completion of jobs. Furthermore Hadoop is fault-tolerant and network intelligent. For companies using Hadoop, this could lead to saving several millions of dollars. As it eliminates the need to monitor traffic, and invest in more servers to achieve better load balancing.

After demonstrating, the prospects of the modified reduce task scheduler, we have set forth two future directions. Firstly, a proposal to introduce a rejection factor that imposes control to ensure uniform reduces task workload among all TaskTracker nodes. As a result, if a TaskTracker is rejected more than a predetermined number of times, the scheduler overrules data locality and ensures uniform workload distribution. Secondly, there may be situations where achieving rack locality is more essential than achieving data locality. Introduction of a locality factor that keeps track of the distance through which the information has to travel, the inherent congestion in the network lines chosen and if transporting more bytes can result in better MapReduce execution time. Based on the computed locality factor, the reduce task scheduler can make more intelligent decisions.

References

1. The New York Times archive article http://open.blogs.nytimes.com/2007/11/01/self-service-prorated super-computing-fun.
2. Making of the New York Time archive called TimesMachine http://open.blogs.nytimes.com/2008/05/21/the- new-york-times-archives-amazon-web-services-timesmachine/ .
3. Hadoop – The Definitive Guide by Tom White. Pro Hadoop by Jason Venner.
4. Website - http://hadoop.apache.org/.
5. Distributed and Cloud Computing by K. Hwang, G. Fox and J. Dongarra.
6. Getting Started with Hadoop http://wiki.apache.org/hadoop/GettingStartedWithHadoop.
7. Getting Started with Maven - http://maven.apache.org/guides/getting-started/.
8. Hadoop operations by Eric Sammer
9. S. Seo, I. Jang, K. Woo, I. Kim, J. Kim, S. Maeng, ”HPMR: Prefetching and Pre-Shuffling in Shared MapReduce Computation Environment,” CLUSTER, 2009.
10. A Szalay, A. Bunn, J. Gray, I. Foster and I. Raicu, the Importance of DataLocality in Distributed Computing Applications,” NSF Workflow Workshop, 2006.
11. M. Zaharia, D. Borthakur, J. S. Sarma, K Elmeleegy, S Shenker, and I. Stoica,”Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling,” EuroSys 2012.
12. M. Zaharia, A. Konwinski, A. Joseph, R. Katz, I. Stoica, ”Improving Mapreduce Performance in Heterogeneous Environments,” *OSDI*, 2008.
13. Amazon Elastic MapReduce, http://aws.amazon.com/elasticmapreduce/.
14. P. C. Chen, Y. L. Su, J. B. Chang, and C. K. Shieh, ”Variable- Sized Map and Locality-Aware Reduce on Public-Resource Grids,” *GPC*, 2010.
15. S. Chen and S. W. Schlosser, ”MapReduce Meets Wider Varieties of Applications,”*IRP-TR-08-05, Intel Research*, 2008.
16. J. Dean and S. Ghemawat, ”Mapreduce: simplified data processing on large clusters,” *OSDI*, 2004.
17. http://download.oracle.com/javase/6/docs/.
18. Mohammad Hammoud and Majd F. Sakr,” Locality-Aware Reduce Task Scheduling for MapReduce” CLOUDCOM '11 Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science.